Digital Technology

PyGame Zero

Introduction

Create a Presentation

Version 3
2021

Barry Butler
bbutl58@eq.edu.au

ROBOCOAST
Sunshine Coast Robotics
www.robocoast.tech

Coolum
State High School
Care • Respect • Excellence

Content

You must install Python and Mu before you can begin. Your teacher may have a USB drive you can use or install them yourself:

1. Python from https://www.python.org/downloads/
2. Mu IDE from https://codewith.mu/en/download

A. PyGame Drawing

Using the Mu Editor

- Click on the mode button (top left) and change the mode to PyGame Zero.
- ***All code files must be saved in the mu_code folder.***
- ***All images must be saved in the mu_code\images folder.***

PyGame Zero Basic Code

All programs require some basic code to put a window on the screen, and draw in that window. For PyGame Zero it is:

```
WIDTH = 800
HEIGHT = 600            #screen dimensions

def draw():
    pass                #remove this when you write some code
```

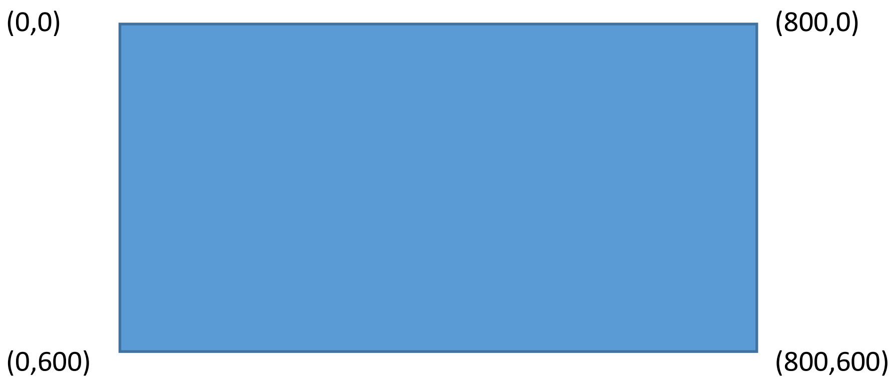
- The screen height and width are **constants** – they do not change.
- The code *def draw():* is a **function** – lines of code that perform a specific task. Notice how code is indented within the function.

Comments

Note the comment – starting with #. Multi-line comments start and end with """.

PyGame Zero Window Coordinates

The top left corner of the window is the origin (0,0). The horizontal coordinate (x) is zero, and the vertical coordinate is zero.



Clear the Window and Set the Window Color

In PyGame Zero, the draw() function is called 60 times every second to redraw the window. Every time the window is redrawn we must clear everything off it and fill it with a color.

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()                #clear screen
    screen.fill('lightskyblue')   #fill with a color
```

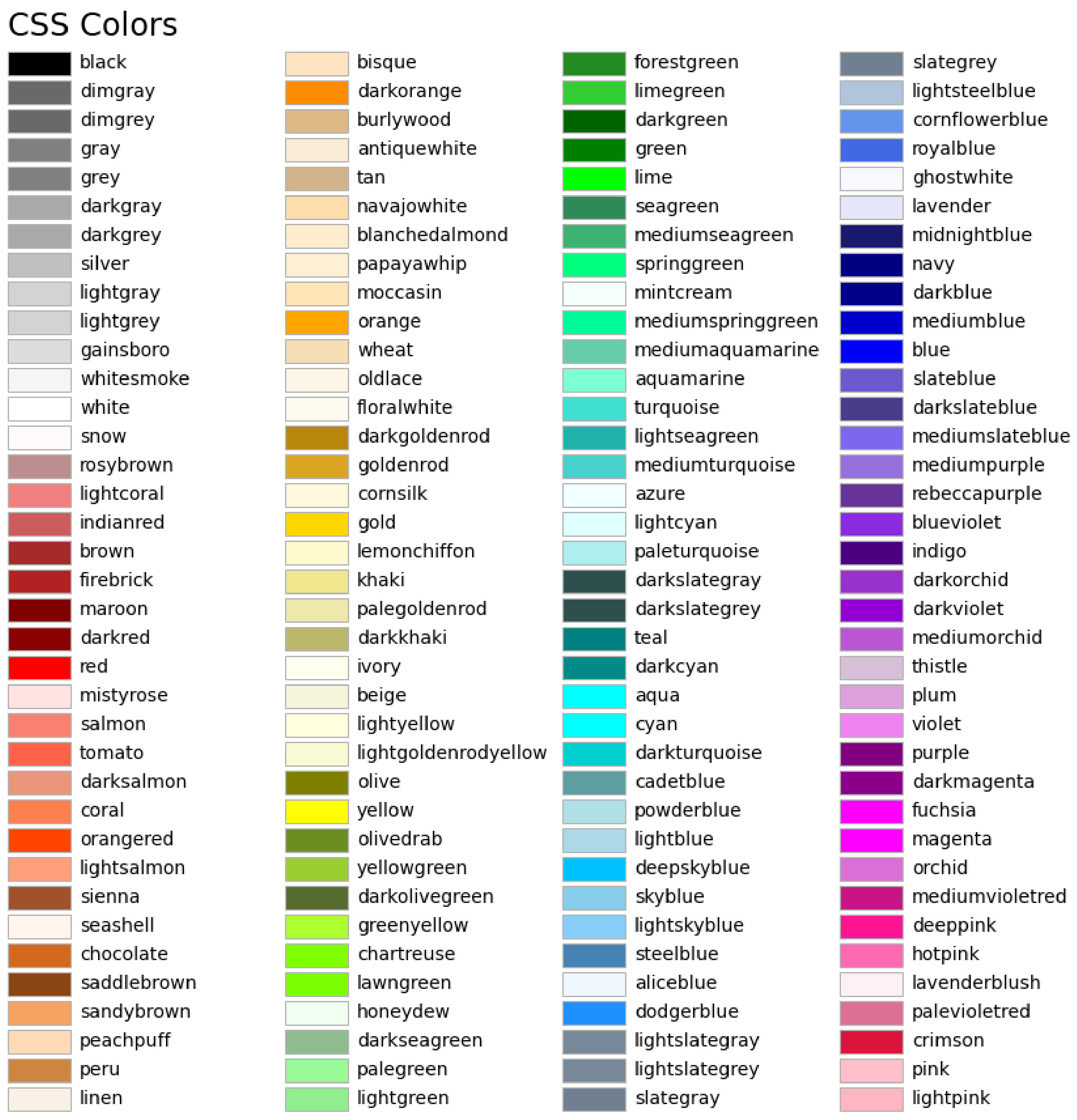
You have already written the code in **black**.

New code to write is shown in **blue**.

Don't write the # comments.

You can only have one draw() function in your program

Python has lots of standard colors. Try some out! Make sure you put the names in quotes, e.g. 'lime'.



CSS Colors

black	bisque	forestgreen	slategrey
dimgray	darkorange	limegreen	lightsteelblue
dimgrey	burlywood	darkgreen	cornflowerblue
gray	antiquewhite	green	royalblue
grey	tan	lime	ghostwhite
darkgray	navajowhite	seagreen	lavender
darkgrey	blanchedalmond	mediumseagreen	midnightblue
silver	papayawhip	springgreen	navy
lightgray	moccasin	mintcream	darkblue
lightgrey	orange	mediumspringgreen	mediumblue
gainsboro	wheat	mediumaquamarine	blue
whitesmoke	oldlace	aquamarine	slateblue
white	floralwhite	turquoise	darkslateblue
snow	darkgoldenrod	lightseagreen	mediumslateblue
rosybrown	goldenrod	mediumturquoise	mediumpurple
lightcoral	cornsilk	azure	rebeccapurple
indianred	gold	lightcyan	blueviolet
brown	lemonchiffon	paleturquoise	indigo
firebrick	khaki	darkslategray	darkorchid
maroon	palegoldenrod	darkslategrey	darkviolet
darkred	darkkhaki	teal	mediumorchid
red	ivory	darkcyan	thistle
mistyrose	beige	aqua	plum
salmon	lightyellow	cyan	violet
tomato	lightgoldenrodyellow	darkturquoise	purple
darksalmon	olive	cadetblue	darkmagenta
coral	yellow	powderblue	fuchsia
orangered	olivedrab	lightblue	magenta
lightsalmon	yellowgreen	deepskyblue	orchid
sienna	darkolivegreen	skyblue	mediumvioletred
seashell	greenyellow	lightskyblue	deeppink
chocolate	chartreuse	steelblue	hotpink
saddlebrown	lawngreen	aliceblue	lavenderblush
sandybrown	honeydew	dodgerblue	palevioletred
peachpuff	darkseagreen	lightslategray	crimson
peru	palegreen	lightslategrey	pink
linen	lightgreen	slategray	lightpink

Draw Lines

Draw a line using the coordinates of the two ends. This code is placed in the *draw()* function.

```
def draw():
    screen.clear()
    screen.fill('lightskyblue')

    screen.draw.line((0,0), (750,550), 'black')
```

The line drawing function has three **parameters** (bits of information it needs to draw the line), separated by **commas**.

- Parameter 1: *(0,0)* - the (x,y) coordinates of the start point
- Parameter 2: *(750,550)* - the (x,y) coordinates of the end point
- Parameter 3: 'black' - the drawing color

Add more instructions to draw lines anywhere on the screen.

Draw Circles

Draw filled and unfilled circles using the center coordinates and the radius.

```
def draw():
    screen.clear()
    screen.fill('lightskyblue')

    screen.draw.line((0,0), (750,550), 'black')

    screen.draw.circle((300,200), 100, 'yellow')
    screen.draw.filled_circle((600,450), 50, 'red')
```

The line drawing function has three **parameters** (bits of information it needs to draw the line), separated by **commas**.

- Parameter 1: *(300,200)* - the (x,y) coordinates of the center
- Parameter 2: *100* – the circle radius
- Parameter 3: 'red' - the drawing color

Add more instructions to draw circles anywhere on the screen

Draw Rectangles

To draw filled and unfilled rectangles we first define the size of the rectangle, then call the function to draw the rectangle.

```
def draw():
    screen.clear()
    screen.fill('lightskyblue')

    screen.draw.line((0,0), (750,550), 'black')

    screen.draw.circle((300,200), 100, 'yellow')
    screen.draw.filled_circle((600,450), 50, 'red')

    screen.draw.rect(Rect((20,100),(360,400)), 'purple')
    screen.draw.filled_rect(Rect((0,400),(300,100)), 'darkblue')
```

A rectangle is defined by the coordinate of the top left (20,100) and a tuple containing the width and height (360,400).

Add more instructions to draw rectangles anywhere on the screen

Our code is getting quite long. To make things shorter we will use

• • •

Keep what you've got and add in the new code in the position shown.

Draw Text

Look on your PyGame Zero Cheat Sheet and find the code for drawing text.

```
def draw():
    . . .

    screen.draw.text("Coding is Cool!", (350,30), color='black', fontsize=60)
```

Add more instructions to draw text anywhere on the screen.

Draw Images

To draw images, we create them as an Actor.

```
alien = Actor('alien', center = (400,200))

def draw():
    . . .

    alien.draw()
```

The order of your code is important.

Make sure the new code in blue looks **exactly** like this.

Look in the **Images folder** for more images and add more instructions to draw these anywhere on the screen.

Find images with transparent backgrounds using google (e.g. rock, spaceship) and copy them into the images folder.

The names of images must only be typed in **lower case** and placed in **single** or **double quotes**.

B. Fixing PyGame Coding Errors

Almost all error messages will give you the **line number** on which it occurred. Sometimes you will need to look at the **previous line**, because the error is at the end of the line (especially a missing colon [:] or closing bracket)

There are a number of basic coding errors that are commonly made in Python. You have probably made some of these already when you were coding section A.

- **Missing Code Errors**. You must copy the examples in this book in exactly the way they have been written. Don't leave anything out. As you work more with Python you will begin to see the patterns.

- **Spelling Errors**. Python and PyGame Zero have built-in words that need to be spelt correctly or nothing will work. Python is case-sensitive.

- **Upper and lowercase Errors**. Almost everything is written in lower case, but some things are in uppercase and must be written in the correct case. Words may be joined by an underscore.

- **Syntax Errors**. Syntax is the grammar of the Python language – commas, brackets, semi-colons and operators (=, ==, >, < etc) in the right places, where they are expected to be.

- **Indentation Errors**. The code after semi-colons is ***always*** indented. Always use the Tab key on your keyboard to indent the code. Code should be aligned to the vertical lines in Mu.

  Some code is always hard up against the left margin. Statements starting with ***import*** and ***def*** are some of these.

- **Assignment and Comparison Errors**. Values are assigned to variables using the **=** operator. Two values are compared using the **==** operator.

Once your code runs, you may also have two other errors:

- **Runtime Errors**. The code will stop because there is an error. These are mostly due to variable values having values that are out-of-bounds.

- **Logic Errors**. Even if your code runs well, it may not do the things you want it to. These are logic or sequencing errors. You are not giving the command statements in an order that makes things work the way you want them to. For example, in the draw() function – writing text on the screen, then clearing the screen and filling it with a colour. The text will never appear.

C. PyGame Variables and Movement

Variables enable us to store values (e.g. text and numbers).

The value of a variable can change.

They must have a descriptive name so you can recognise them.

The variable is always assigned with the equal sign, followed by the value of the variable. Use descriptive names.

Variable Naming Conventions

- **Lowercase** words separated by **underscores**
- Begin with a letter of the alphabet

Types of Data

Variables are given a data type when they are assigned (e.g. i = 1). There are four basic data types:

• **Integer**	e.g. 1, -120, 3000
• **Floating point** (Decimal)	e.g. 1.45, -12.564, 4000.1
• **String** (Text)	e.g. “First name”, ‘OK’
• **Boolean**	True, False (with capital T and F)

Moving Lines

Instead of using a fixed value in a function we can use a **variable**. Then we can change the value of the variable to create movement. First, start a new file.

Start a new file for this section.

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')
```

Then write the code to move a line across the screen. We declare the variable before the *draw()* function. The *update()* function changes the value of the variable to create movement.

```
WIDTH = 800
HEIGHT = 600

start_x = 20

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    screen.draw.line((start_x,20), (400,550), 'black')

def update():
    global start_x
    start_x += 1                    #add 1 to the value of line_x
```

Add other variables for the start and end coordinates of the line and change them too (note: -= subtracts a value).

Global vs Local Variables

All variables are **local** to a function by default. That is, they only exist inside a specific function and cannot be used in other functions.

The variable ***start_x*** is created outside a function, and can be used in all functions. It is called a **global** variable.

To change a global variable we must put a ***global*** statement in the first line of the function code.

Moving Circles

To move a circle let's use three variables, for the x value, y value and the radius.

```
WIDTH = 800
HEIGHT = 600

start_x = 20
circle_x = 600
circle_y = 450
circle_radius = 50


def draw():
    screen.clear()
    screen.fill('lightskyblue')

    screen.draw.line((start_x,20), (400,550), 'black')
    screen.draw.filled_circle((circle_x,circle_y), circle_radius, 'red')

def update():
    global start_x, circle_x, circle_y, circle_radius

    start_x += 1
    circle_x -= 1
    circle_y -= 1
    circle_radius += 1
```

Notice how similar things are put together

The global variables

The drawing code

The variable updating code

Moving Images

Actors and their images are **objects** which have variables built into them (called **properties**). Images have x and y properties that can be changed.

```
. . .
alien = Actor('alien', center = (10,10))

def draw():
    . . .
    alien.draw()

def update():
    . . .
    alien.x += 2
    alien.y += 1
```

Make sure you put things in their correct place

The global variable

The drawing code

The variable updating code

Add more Actors with their images, and move them.

D. PyGame Loops

Start a new file.

Start a new file for this section.

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')
```

Loops provide an easy way of repeating a series of steps, without duplicating the code. For example, we can draw a series of circles:

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    radius = 10
    for i in range(10):
        screen.draw.circle((400,300), radius, 'red')
        radius += 10
```

The circle will be drawn 10 times. The variable `i` is the loop counter – it keeps track of how many times the loop has executed.

The variable radius is increased every time the loop is executed. We must set the initial value of a variable before the loop, then change the value with each iteration of the loop.

Draw a series of lines:

Remember

• • •

means you keep what you've got and add in the new code in the position shown.

```
def draw():
    . . .

    x = 50
    for i in range(20):
        screen.draw.line((x,20), (x,200), 'black')
        x += 20
```

For Loop with Three Parameters

The *For* Loop can actually take three parameters – the **start value**, the **end value** and the **step value**.

```
def draw():
    . . .

    for y in range(40,600,40):
        screen.draw.line((500,y), (750,y), 'yellow')

    for y in range(300,500,30):
        screen.draw.text('Hello There', topleft=(600,y), color='black', fontsize=20)
```

Use loops to draw other things on your screen: regular shapes, rows of circles around the border etc.

E. Conditional Statements

Start a new file.

Start a new file for this section.

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')
```

Conditional statements are used to make **decisions**, creating different pathways depending on variable values. They have three key words: **if**, **elif** and **else**.

If a conditional statement is true, a set of instructions is executed. If the statement is false, then the instructions are not executed or the instructions in the **else** statement are executed

Example 1 – two choices (if-else)

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    for i in range(20):
        if i < 10:                 #conditional statement (is i < 10?)
            col = 'red'              #color if statement is True
        else:
            col = 'yellow'           #color if statement is False


        screen.draw.filled_circle((300,300+i*3), 70, col)
```

Example 2 – three choices (if-elif-else)

```
def draw():
    . . .

    x = 50
    for i in range(20):
        if i % 3 == 0: col = 'black'
        elif i % 2 == 1: col = 'red'
        else: col = 'yellow'
        screen.draw.line((x,20), (x,200), col)
        x += 20
```

The **%** operator is the **modulus** operator.

The modulus is the **remainder** of the division. In this example we can tell if a number is **even** or **odd**.

- To compare values we use **== (or <, >, <=, >=). !=** means not equal to (or use the keyword ***not***)
- To assign a value to a variable we use **=**.

Example 3 – Write text depending on the score

```
WIDTH = 800
HEIGHT = 600

score = 0

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    if score >= 100:
        s = 'You Win!'
        c = 'black'
    else:
        s = 'You Will Win Soon!'
        c = 'red'
    screen.draw.text(s, midtop=(600,10), color=c, fontsize=40)

def update():
    global score
    score += 1
```

Make sure you put things in their correct place

The global variable

The comparison and drawing code

The variable updating code

Use Conditional Statements with Actors

Start a new file.

Start a new file for this section.

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')
```

Now we create and draw an alien and give it a variable called **visible**.

```
WIDTH = 800
HEIGHT = 600

alien = Actor('alien', center = (100,100))
alien.visible = True

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    if alien.visible:
        alien.draw()
```

Change the statement to `alien.visible = False`. What happens?

Use Conditional Statements to Limit Movement

We often want to limit the movement of an Actor to a portion of the screen (or make sure it does not go outside the screen).

Let's make an alien move on the screen.

```
WIDTH = 800
HEIGHT = 600

alien = Actor('alien', center = (100,100))
alien.visible = True
alien.x_speed = 2                   #pixels to move each time
alien.y_speed = 3

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    if alien.visible:
        alien.draw()

def update():
    alien.x += alien.x_speed        #change x and y position
    alien.y += alien.y_speed
```

Now let's fix the problem of going off the screen. We use a double conditional statement to check whether the Actors' x or y coordinates are outside the screen.

```
def update():
    alien.x += alien.x_speed
    alien.y += alien.y_speed

    if (alien.x <= 0) or (alien.x >= WIDTH):     #if x off screen
        alien.x_speed = -alien.x_speed           #reverse direction

    if (alien.y <= 0) or (alien.y >= HEIGHT):    #if y off screen
        alien.y_speed = -alien.y_speed
```

or means ***either*** of the statements must be True

and means ***both*** statements must be True

Test for Collisions

A fundamental event in games is collisions between sprites. We will use a rock image for the second sprite. If you do not have a rock image in your **Images** folder, download one from the internet or change it for another you do have.

```
WIDTH = 800
HEIGHT = 600

alien = Actor('alien', center = (100,100))
alien.visible = True
alien.x_speed = 3
alien.y_speed = 2

rock = Actor('rock', center = (400,300))   # 'rock' is the name of an image file

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    if alien.visible:
        alien.draw()
    rock.draw()

def update():
    alien.x += alien.x_speed
    alien.y += alien.y_speed

    if alien.colliderect(rock):                  #if collide with the rock
        alien.visible = False

    if (alien.x <= 0) or (alien.x >= WIDTH):
        alien.x_speed = -alien.x_speed

    if (alien.y <= 0) or (alien.y >= HEIGHT):
        alien.y_speed = -alien.y_speed
```

F. PyGame Lists

Start a new file.

Start a new file for this section.

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')
```

Lists make it easy to store and access a sequence of data. Lists are a **comma-separated** sequence surrounded by **square brackets**, and assigned to a variable. The variable then has the data type **list**. The data in a list can be any type, including mixed types.

To create a list, declare it (use a plural name):

```
numbers = [1, 2, 3, 4, 5]

measurements = [1.2, 3.7, 4.5, 8.7, 10.6]

answers = [False, True, True, False]

my_records = ['Bob', 24, 'Havana Rd', 4567, True, False]
```

Access List Values

List values are accessed by indexing the list variable. The first item of the list has an index of [0].

```
WIDTH = 800
HEIGHT = 600

names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    s = 'First is ' + names[0]
    screen.draw.text(s, topleft=(100,50), color='red', fontsize=60)

    s = 'Third is ' + names[2]
    screen.draw.text(s, topleft=(100,150), color='red', fontsize=60)

    s = 'Last is ' + names[-1]
    screen.draw.text(s, topleft=(100,250), color='red', fontsize=60)
```

Loop Through All List Values (Method 1)

The **for** loop can be used to access all items in a list. Replace the draw() code with this:

```
WIDTH = 800
HEIGHT = 600

names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    y = 50
    for name in names:
        screen.draw.text(name, topleft=(600,y), color='yellow', fontsize=40)
        y += 50
```

It is common to use a plural description for a list (e.g. names)

The loop variable is the singular of the same description (e.g. name)

Loop Through All List Values (Method 2)

The **for** loop can also be used to access items by **index**.

```
WIDTH = 800
HEIGHT = 600

names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

def draw():
    screen.clear()
    screen.fill('lightskyblue')

    y = 50
    for name in names:
        screen.draw.text(name, topleft=(600,y), color='yellow', fontsize=40)
        y += 50

    x = 40
    for i in range(len(names)):
        screen.draw.text(names[i], topleft=(x,500), color='red', fontsize=30)
        x += 100
```

The **len()** function returns the length of the list (number of items). Each list item is accessed using the loop variable `i` (e.g. `dogs[i]`)

G. PyGame Mouse and Keyboard Events

Start a new file.

Start a new file for this section.

```
WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.fill('lightskyblue')
```

Respond to Mouse Clicks and Continuous Keyboard Presses (start new file)

Responding to mouse clicks is easy in PyGame Zero. Let's click on an alien.

```
WIDTH = 800
HEIGHT = 600

alien = Actor('alien', center = (400,300))

def draw():
    screen.clear()
    screen.fill('lightblue')

    alien.draw()

def on_mouse_down (pos, button):          #respond to mouse clicks
    if button == mouse.LEFT:
        if alien.collidepoint(pos):       #if the mouse pos collides with the image
            if alien.image == 'alien':
                alien.image = 'alien_hurt'
                alien.angle = 180
            else:
                alien.image = 'alien'
                alien.angle = 0

def update():                             #respond to continuous keyboard presses
    if keyboard.right:
        alien.x += 1
        alien.angle = 270
    elif keyboard.left:
        alien.x -= 1
        alien.angle = 90
```

If we only wanted to respond to separate key presses (rather than holding down a key) we use the on_key_down() function.

```
def on_key_down(key):
    global choice
    if key == keys.UP:
        alien.y -= 5
    elif key == keys.DOWN:
        alien.y += 5
```

H. Create a PyGame Presentation

PyGame Zero can be used to create interactive presentations. Screen elements may include:

- Rectangles of different colour, with text or images inside them
- Images
- Text – of different sizes and/or fonts
- Sprites – little images that can be clicked on and then do something.
- Buttons – made of a rectangle and a sprite
- Responses to keyboard keys
- Responses to mouse clicks (e.g. inside a button or on a sprite or image)

H1. PyGame Zero Basic Code

The following code is used for all PyGame Zero applications. Type this code into the Mu editor (in **PyGame Zero mode**).

```
WIDTH = 800                         #set screen dimensions
HEIGHT = 600

def draw():
    screen.clear()                  #clear screen and fill with colour
    screen.fill('deepskyblue')

def on_key_down(key):
    pass                            #required because there is no code here yet

def on_mouse_down(pos, button):
    pass
```

Notes:

1. The width and height sets up a window with those dimensions
2. There are three standard functions that the program automatically calls:
    a. draw() – draws objects on the screen
    b. update() – gets arrow key inputs and changes the position of objects before they are drawn
    c. on_key_down() – makes things happen as a result of pressing keyboard keys

EACH TIME YOU ENTER CODE, CLICK THE “PLAY” BUTTON TO TEST IT, AND FIX ANY ERRORS.

H2. Create a Rectangle to hold Text

Almost all programs use rectangles of different colors to divide the screen into several parts. It is easiest to work with rectangles if they are declared as constants at the beginning of the program. They can easily be changed later.

```
WIDTH = 800
HEIGHT = 600

TEXTRECT = Rect((20,100),(360,400))  #(x,y),(width,height)

def draw():
    screen.clear()
    screen.fill((0, 250, 250))

    screen.draw.filled_rect(TEXTRECT, 'lightskyblue')
```

H3. Create and Draw Images

Images are created as actors. You can have as many images as you like.

```
sheep1 = Actor("sheep1", topleft = (350,120))

def draw():
    screen.clear()
    screen.fill((0, 250, 250))

    sheep1.draw()
    screen.draw.filled_rect(TEXTRECT, 'lightskyblue')
```

H4. Write a Title

Titles are usually centred in the window, a little experimentation is required to get the coordinates correct.

```
def draw():
    screen.clear()
    screen.fill((0, 100, 250))

    screen.draw.text("Sheep are Cool!", (260,30), color='yellow', fontsize=60)
    sheep1.draw()
    screen.draw.filled_rect(TEXTRECT, 'lightskyblue')
```

H5. Write Some Text in the Rectangle

The easiest way to write text is to create a list with each item a different line of text. Each line of text is separated by a comma.

In the draw() function, loop through the list to draw each line of text on the screen. Use the position of the rectangle to position the text.

```
page1_text = ['This is how it all started…',
                'Someone caught a wild sheep',
                'and put it into pen.',
                'They fed it grass and oats.',
                'When it was big and fat they',
                'killed it and ate it'
              ]

def draw():
    . . .
    screen.draw.filled_rect(TEXTRECT, 'lightskyblue')

    y = 120
    for line in page1_text:
        screen.draw.text(line, (40,y), color='black', fontsize=30)
        y += 35
```

Notice that the first line of text starts at position y = 120. Each iteration of the loop adds 35 to y. So each line of text is spaced 35 pixels apart. Change these values to alter where the text is on the screen and the spacing between the lines.

H6. Create Multiple Pages

Create constants for each page, and a variable to hold the value of the current page. In the draw() function, use 'if-elif' statements to separate the elements that are drawn on each page. Locate elements common to all pages before or after the 'if-elif' statements.

Note: to shift many lines of code to the right, select all the lines and press Tab. To move them to the left, select all the lines and press Shift-Tab.

```
PAGE1 = 0                     #page constants
PAGE2 = 1
PAGE3 = 2
PAGE4 = 3
page = PAGE1                  #page variable – holds the value of the current page

def draw():
    screen.clear()
    screen.fill((0, 100, 250))
    screen.draw.text("Sheep are Cool!", (260,30), color='yellow', fontsize=60)
    screen.draw.filled_rect(TEXTRECT, 'lightskyblue')

    if page == PAGE1:
        sheep1.draw()

        y = 120
        for line in page1_text:
            screen.draw.text(line, (40,y), color='black', fontsize=30)
            y += 35
    elif page == PAGE2:
        pass
    elif page == PAGE3:
        pass
    elif page == PAGE4:
        pass
    screen.draw.text(str(page), (WIDTH/2,HEIGHT-50), color='black', fontsize=30)
```

The last line draws the page number on the screen. Draw all the elements required for each page inside the elif statement for that page.

H7. Draw Navigation Sprites

Add the arrows (or any other sprite) as an Actor. Using the WIDTH and HEIGHT constants to position these is a great idea because they will stay in the correct places regardless of the window size.

```
sheep1 = Actor("sheep1", topleft = (350,120))
left_arrow = Actor("leftarrow", topleft = (30,HEIGHT-70))
right_arrow = Actor("rightarrow", topleft = (WIDTH-90,HEIGHT-70))
```

Draw the arrows on the window:

```
def draw():
    . . .
    screen.draw.text(str(page), (WIDTH/2,HEIGHT-50), color='black', fontsize=30)
    left_arrow.draw()
    right_arrow.draw()
```

H8. Use Arrow Keys to Move between Pages

We use the on_key_down() function to capture when keyboard keys are pressed, and navigate between pages.

```
def on_key_down(key):
    global page                          #required to change a global variable
    if key == keys.RIGHT:
        if page < PAGE4:
            page += 1
    elif key == keys.LEFT:
        if page > PAGE1:
            page -= 1
```

H9. Mouse Click on the Sprites to Move between Pages

To respond to mouse clicks, we test whether the mouse x and y coordinates are within the sprite rectangle. We put the code in the on_mouse_down() function.

```
def on_mouse_down(pos, button):
    global page
    if button == mouse.LEFT:
        if left_arrow.collidepoint(pos):
            if page > PAGE1:
                page -= 1
        elif right_arrow.collidepoint(pos):
            if page < PAGE4:
                page += 1
```